Introducing CFNetwork

Session 805

Introducing CFNetwork

Becky Willrich
Application Frameworks

Introducing CFNetwork

- CFNetwork overview
- Basic concepts
- CFSocketStream
- CFHTTPMessage and CFHTTPStream
- CFNetServices
- Demo

CFNetwork Overview

CFNetwork Overview

- What is it?
- How does it fit in?
- What does it provide?
- When should you use it?
- Where is it available?

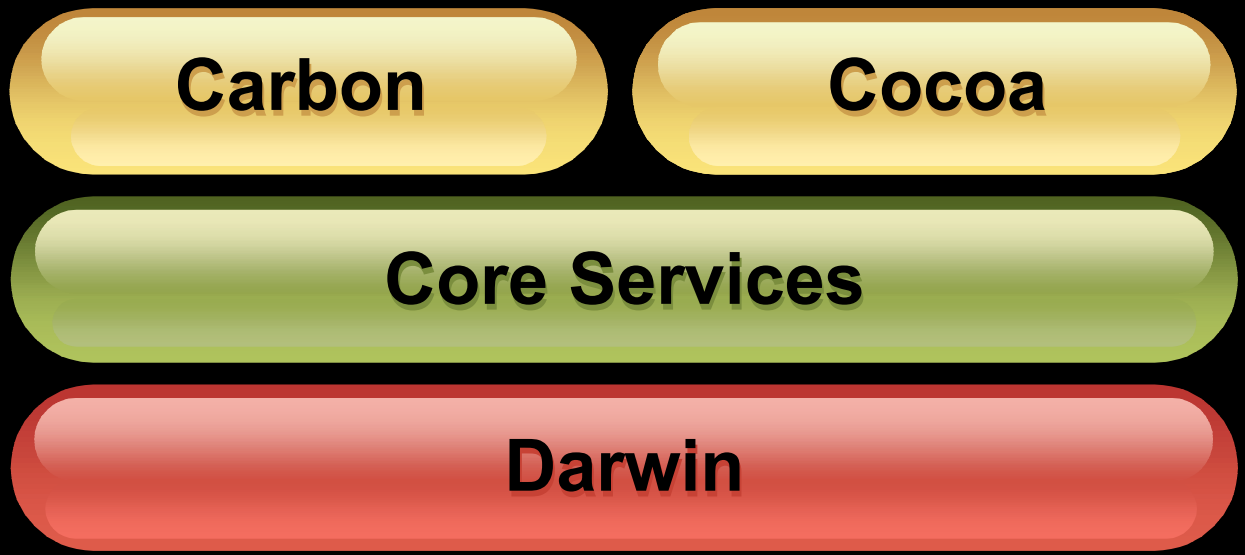
What Is CFNetwork?

- Library of abstractions for network protocols
  - Sockets
  - HTTP
  - Rendezvous
- APIs in the style of CoreFoundation

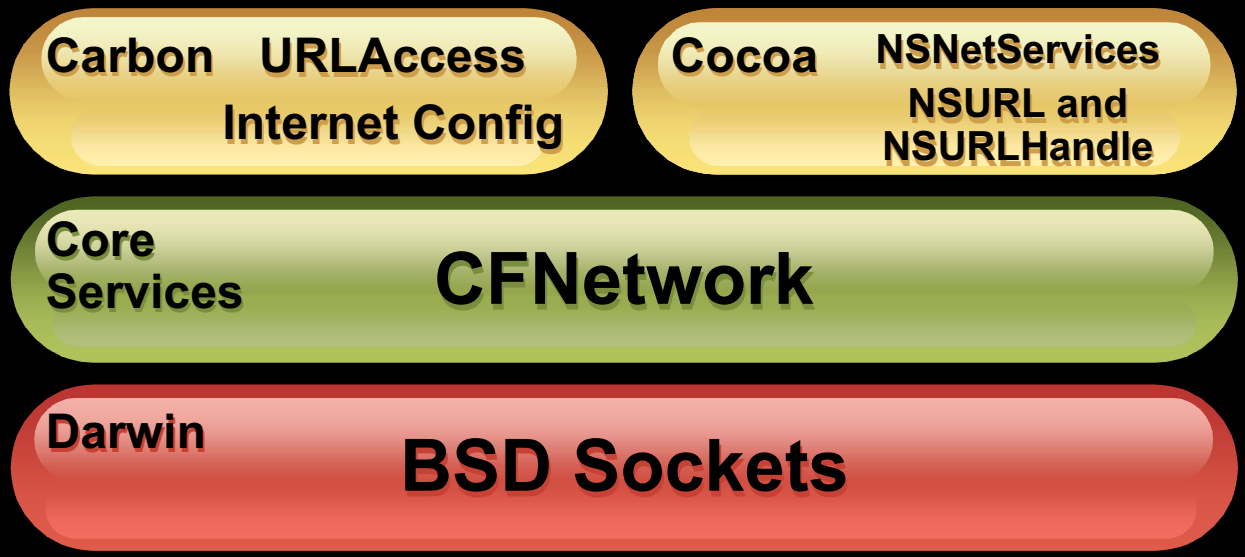
Goals of CFNetwork

- Expose the full power of the underlying protocol
- Strong integration with existing libraries and paradigms
    - Especially the run loop!
- Little to no overhead compared to raw sockets
- Will not protect you from the details

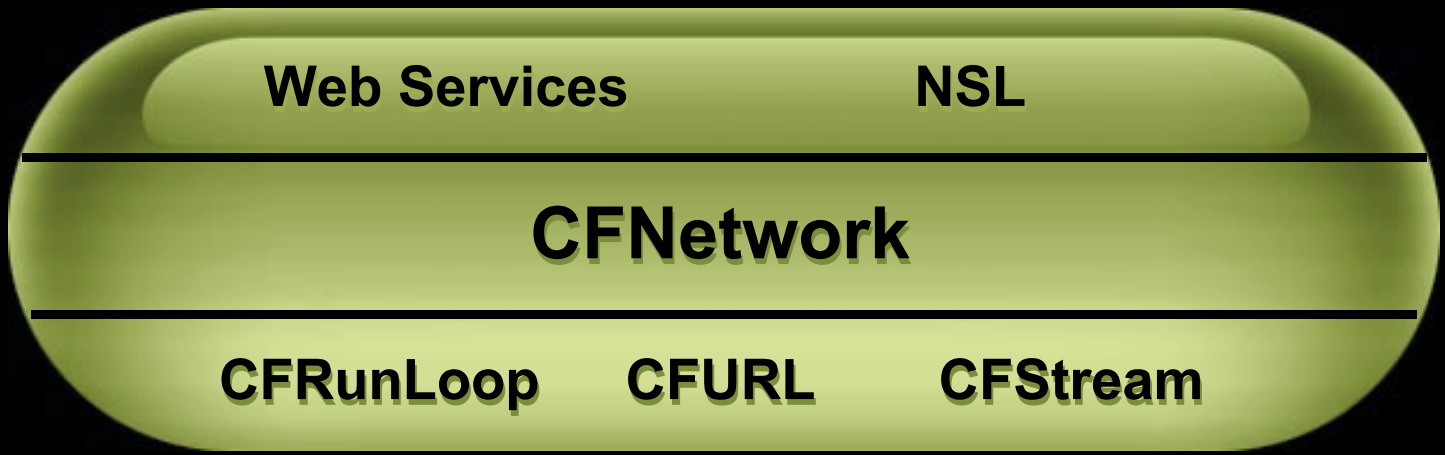
How Does It Fit In?



How Does It Fit In?



How Does It Fit In?



How Does It Fit In?

- Part of CoreServices.framework
- Builds on BSD sockets plus CoreFoundation
- Used by higher level frameworks
    - NSURL, NSL, Web Services
- Used directly by many apps
    - Mail, Software Update, Sherlock, iPhoto

What Does CFNetwork Provide?

- Common socket abstractions (SSL/TLS, SOCKS)
  - CFSocketStream
- HTTP/1.1 engine
  - CFHTTPMessage
  - CFHTTPStream
- Network service registry and discovery
  - CFNetService
  - CFNetServiceBrowser

When Should You Use CFNetwork?

- Use CFNetwork if you need a low-level, high-performance framework
  - Working directly on sockets
  - Need control of when bytes are read/written
  - Need detailed control of HTTP
  - Want to link low
  - Need new features only available in CFNetwork
- Use the higher APIs instead if possible

When Should You Use CFNetwork?

- Use CFNetwork if raw sockets is too low:
  - Want run loop integration
  - Want to avoid managing select()
  - Want simple TLS/SSL use
  - Want an “object” abstraction (reference counting)

Where Is CFNetwork Available?

- 10.1 forward only
- Many features added in Jaguar
    - SOCKS support
    - CFNetServices
    - Persistent HTTP/1.1 connections

Basic Concepts

Basic Concepts

- Two basic concepts from CoreFoundation
  - Streams
  - Run loops
- We'll cover the basics . . .
- . . . but go to Session 808: Managing I/O!

Streams

- One-way byte stream to some source or destination
- API similar to POSIX file descriptors
  - Open, read/write, close . . .
- Two true CFTypes
  - CFReadStream
  - CFWriteStream
- Other “types” are just special cases of CFRead/WriteStreams
  - CFSocketStream, CFHTTPStream, . . .

Stream Properties

- Streams carry properties
  - Describe how the stream behaves
  - Information discovered by the stream
- Set properties to configure a stream
- Query properties to find out about a stream

Using Properties

- Get/Set properties by name
  - Names are CFStrings
  - Values are CFTypes
- Streams can refuse properties
  - If it doesn't recognize the property
  - If the value is invalid or inappropriate
  - If the stream has been opened
- If refused, CopyProperty() returns NULL, SetProperty() returns FALSE

Properties—Example

```
CFHTTPMessageRef headers =
    CFReadStreamCopyProperty(httpStream,
        kCFStreamPropertyHTTPResponseHeaders);

CFReadStreamSetProperty(stream,
    kCFStreamPropertySocketSecurityLevel,
    kCFStreamSocketSecurityLevelNegotiatedSSL);

CFReadStreamSetProperty(stream,
    kCFStreamPropertyHTTPProxy,
    proxyDict);
```

Run Loops

- Also a CFType—CFRunLoop
- Abstraction for an event loop for arbitrary event sources
- Typical sources:
  - User events
  - Mach messages/IPC
  - Timers
  - Sockets
- Exactly one run loop per thread

Run Loops

- Run loop watches its sources
  - If something interesting happens to a source . . .
    - . . the run loop calls a callback
    - . . the callback finishes
    - . . the run loop goes back to watching
- So you can manage multiple inputs on one thread

Run Loops and Clients

- Many CFNetwork objects take clients with callbacks
  CFReadStreamSetClient(stream, myClient, myCallBack);
- Then you schedule on a run loop
  CFReadStreamScheduleWithRunLoop(stream, CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
- You may schedule on multiple run loops if you wish, but . . .
- Protecting the scheduled objects is your responsibility

Run Loops and Clients

- As the run loop runs, the CFNetwork object detects changes
- Those changes trigger your callback
- It is your responsibility to ensure the run loop is running
    - Both the Cocoa and Carbon UI frameworks do this for you
- If the run loop is not running, you will never get your callback

Run Loops and Clients—Cleaning Up

- CFRelease(stream) is not enough!
- You must call SetClient(NULL) to guarantee you will never be called again
- You should also unschedule from any run loops to avoid wasting CPU

```
CFReadStreamSetClient(stream, 0, NULL, NULL);
CFReadStreamUnscheduleFromRunLoop(stream,
        CFRunLoopGetCurrent(),
        kCFRunLoopCommonModes);
CFRelease(stream);
```

Socket Streams

Socket Streams

- You can create a read stream and write stream pair for a socket
    - By hostname and port number
    - By raw socket
    - By socket signature
- Basic support is in CoreFoundation
- CFNetwork adds a number of configuration properties

Configuring Socket Streams

- Socket properties are defined inside CFNetwork/CFSocketStream.h
- Properties set on the read stream affect the write stream and vice versa
- You may set socket properties on any stream built on sockets (e.g., HTTP)

Socket Properties

- SOCKS proxy settings (kCFStreamPropertySOCKSProxy)
  - Can construct the dictionary yourself
  - Or just pass SystemConfigs
- TLS/SSL level

CFHTTPMessage and CFHTTPStream

CFHTTPMessage and CFHTTPStream

- CFHTTPMessage is an HTTP request or response
- CFHTTPStream performs an HTTP transaction
- Normal usage:
  - Create a CFHTTPMessage for the request
  - Create a CFHTTPStream from the message
  - Open and read from the stream to process the request

CFHTTPMessage

- Represents an HTTP request or response
- Get and set header fields, body
- Two basic usages:
    - Construct a message programmatically
    - Parse out a message from a series of bytes

Constructing a Message

```
CFHTTPMessageRef msg =
    CFHTTPMessageCreateRequest(
        kCFAllocatorDefault, CFSTR("POST"),
        myURL, kCFHTTPVersion1_1);

CFHTTPMessageSetHeaderFieldValue(msg,
    CFSTR("User-Agent"), myUserAgent);

...

CFHTTPMessageSetBody(msg, myPostData);
```

Constructing a Message

```
CFHTTPMessageRef msg =
    CFHTTPMessageCreateRequest(
        kCFAllocatorDefault, CFSTR("POST"),
        myURL, kCFHTTPVersion1_1);

CFHTTPMessageSetHeaderFieldValue(msg,
    CFSTR("User-Agent"), myUserAgent);

...

CFHTTPMessageSetBody(msg, myPostData);
```

Constructing a Message

```
CFHTTPMessageRef msg =
    CFHTTPMessageCreateRequest(
        kCFAllocatorDefault, CFSTR("POST"),
        myURL, kCFHTTPVersion1_1);

CFHTTPMessageSetHeaderFieldValue(msg,
    CFSTR("User-Agent"), myUserAgent);

...

CFHTTPMessageSetBody(msg, myPostData);
```

Parsing Out a Message

```
CFHTTPMessageRef msg =
    CFHTTPMessageCreateEmpty(
        kCFAllocatorDefault, isRequest);

while (numBytes = readNewHTTPData(buf, len))  {

    if (!CFHTTPMessageAppendBytes(msg,
        buf, numBytes)) {
        reportParseError();
    }
}

// Now query msg to see what you've got
```

CFHTTPStream

- Created from a CFHTTPMessage for a request
- Reading will only read the data bytes
- Examine the response's HTTP header by fetching a property
    - kCFStreamPropertyHTTPResponseHeader

CFHTTPStream—Properties

- All the socket properties are available
- HTTP-specific properties in CFNetwork/CFHTTPStream.h
- Set-able:
  - Automatic redirection
  - HTTP proxy settings
  - Persistent connections
- Get-able:
  - Response header
  - Final URL

Using a CFHTTPStream

- Create and configure the stream
- Schedule the stream
- Get the response in your callback
- Clean up when you are done

CFHTTPStream—Creation

```
CFReadStreamRef  createStream() {

    CFHTTPMessageRef myRequest =
        constructRequest();

    CFReadStreamRef httpStream =
        CFReadStreamCreateForHTTPRequest(
            kCFAllocatorDefault, myRequest);

    CFReadStreamSetProperty(httpStream,
        kCFStreamPropertyHTTPShouldAutoredirect,
        kCFBooleanTrue);

    return httpStream;
}
```

CFHTTPStream—Creation

```
CFReadStreamRef  createStream() {

    CFHTTPMessageRef myRequest =
        constructRequest();

    CFReadStreamRef httpStream =
        CFReadStreamCreateForHTTPRequest(
            kCFAllocatorDefault, myRequest);

    CFReadStreamSetProperty(httpStream,
        kCFStreamPropertyHTTPShouldAutoredirect,
        kCFBooleanTrue);

    return httpStream;
}
```

CFHTTPStream—Creation

```
CFReadStreamRef  createStream() {

    CFHTTPMessageRef myRequest =
        constructRequest();

    CFReadStreamRef httpStream =
        CFReadStreamCreateForHTTPRequest(
            kCFAllocatorDefault, myRequest);

    CFReadStreamSetProperty(httpStream,
        kCFStreamPropertyHTTPShouldAutoredirect,
        kCFBooleanTrue);

    return httpStream;
}
```

CFHTTPStream—Scheduling

```
void  scheduleStream(CFReadStreamRef httpStream,
      CFStreamClientContext *myContext) {

      CFOptionFlags events =
            kCFStreamEventOpenCompleted |
            kCFStreamEventHasBytesAvailable |
            kCFStreamEventEndEncountered |
            kCFStreamEventErrorOccurred;

      CFReadStreamSetClient(httpStream, events,
            handleEvent, myContext);

      CFReadStreamScheduleWithRunLoop(httpStream,
            CFRunLoopGetCurrent(),
            kCFRunLoopCommonModes);

      CFReadStreamOpen(httpStream);
}
```

CFHTTPStream—Scheduling

```
void  scheduleStream(CFReadStreamRef httpStream,
      CFStreamClientContext *myContext) {

      CFOptionFlags events =
            kCFStreamEventOpenCompleted |
            kCFStreamEventHasBytesAvailable |
            kCFStreamEventEndEncountered |
            kCFStreamEventErrorOccurred;

      CFReadStreamSetClient(httpStream, events,
            handleEvent, myContext);

      CFReadStreamScheduleWithRunLoop(httpStream,
            CFRunLoopGetCurrent(),
            kCFRunLoopCommonModes);

      CFReadStreamOpen(httpStream);
}
```

CFHTTPStream—Scheduling

```
void  scheduleStream(CFReadStreamRef httpStream,
      CFStreamClientContext *myContext) {

      CFOptionFlags events =
            kCFStreamEventOpenCompleted |
            kCFStreamEventHasBytesAvailable |
            kCFStreamEventEndEncountered |
            kCFStreamEventErrorOccurred;

      CFReadStreamSetClient(httpStream, events,
            handleEvent, myContext);

      CFReadStreamScheduleWithRunLoop(httpStream,
            CFRunLoopGetCurrent(),
            kCFRunLoopCommonModes);

      CFReadStreamOpen(httpStream);
}
```

CFHTTPStream—Scheduling

```
void  scheduleStream(CFReadStreamRef httpStream,
      CFStreamClientContext *myContext) {

      CFOptionFlags events =
            kCFStreamEventOpenCompleted |
            kCFStreamEventHasBytesAvailable |
            kCFStreamEventEndEncountered |
            kCFStreamEventErrorOccurred;

      CFReadStreamSetClient(httpStream, events,
            handleEvent, myContext);

      CFReadStreamScheduleWithRunLoop(httpStream,
            CFRunLoopGetCurrent(),
            kCFRunLoopCommonModes);

      CFReadStreamOpen(httpStream);
}
```

CFHTTPStream—Getting the Response

```
void handleEvent(CFReadStreamRef httpStream,
    CFStreamEventType event, void *myInfo) {

    switch (event) {
    case kCFStreamEventOpenCompleted:
        ...
    case kCFStreamEventEndEncountered:
        ...
    case kCFStreamEventErrorOccurred:
        ...
    case kCFStreamEventHasBytesAvailable:
        ...
    default:
        printf("Should never get here\n");
    }
}
```

CFHTTPStream—Getting the Response

```
void handleEvent(CFReadStreamRef httpStream,
    CFStreamEventType event, void *myInfo) {

    switch (event) {
    case kCFStreamEventOpenCompleted:
        ...
    case kCFStreamEventEndEncountered:
        ...
    case kCFStreamEventErrorOccurred:
        ...
    case kCFStreamEventHasBytesAvailable:
        ...
    default:
        printf("Should never get here\n");
    }
}
```

CFHTTPStream—Getting the Response

```
case kCFStreamEventOpenCompleted:
        printf("Open completed\n");
        break;

case kCFStreamEventEndEncountered:
        printf("All done\n");
        cleanUp(httpStream, myInfo);
        break;

case kCFStreamEventErrorOccurred:
        printf("Error occurred\n");
        handleError(myInfo, CFReadStreamGetError(httpStream));
        cleanUp(httpStream, myInfo);
        break;
...
```

CFHTTPStream—Getting the Response

```
case kCFStreamEventOpenCompleted:
        printf("Open completed\n");
        break;

case kCFStreamEventEndEncountered:
        printf("All done\n");
        cleanUp(httpStream, myInfo);
        break;

case kCFStreamEventErrorOccurred:
        printf("Error occurred\n");
        handleError(myInfo, CFReadStreamGetError(httpStream));
        cleanUp(httpStream, myInfo);
        break;
...
```

CFHTTPStream—Getting the Response

```
case kCFStreamEventOpenCompleted:
        printf("Open completed\n");
        break;

case kCFStreamEventEndEncountered:
        printf("All done\n");
        cleanUp(httpStream, myInfo);
        break;

case kCFStreamEventErrorOccurred:
        printf("Error occurred\n");
        handleError(myInfo, CFReadStreamGetError(httpStream));
        cleanUp(httpStream, myInfo);
        break;
...
```

CFHTTPStream—Getting the Response

```
case kCFStreamEventHasBytesAvailable: {
    UInt8 buf[BUFSIZE];
    CFIndex numBytes;

    if (firstTime) {
        CFHTTPMessageRef responseHeaders =
            CFReadStreamCopyProperty(httpStream,
                kCFStreamPropertyHTTPResponseHeader);
        processHeaders(responseHeaders);
        CFRelease(responseHeaders);
    }

    numBytes = CFReadStreamRead(httpStream, buf, BUFSIZE);
    handleResponseBodyBytes(myInfo, buf, numBytes);
    break;
}
```

CFHTTPStream—Getting the Response

```
case kCFStreamEventHasBytesAvailable: {
    UInt8 buf[BUFSIZE];
    CFIndex numBytes;

    if (firstTime) {
        CFHTTPMessageRef responseHeaders =
            CFReadStreamCopyProperty(httpStream,
                kCFStreamPropertyHTTPResponseHeader);
        processHeaders(responseHeaders);
        CFRelease(responseHeaders);
    }

    numBytes = CFReadStreamRead(httpStream, buf, BUFSIZE);
    handleResponseBodyBytes(myInfo, buf, numBytes);
    break;
}
```

CFHTTPStream—Getting the Response

```
case kCFStreamEventHasBytesAvailable: {
    UInt8 buf[BUFSIZE];
    CFIndex numBytes;

    if (firstTime) {
        CFHTTPMessageRef responseHeaders =
            CFReadStreamCopyProperty(httpStream,
                kCFStreamPropertyHTTPResponseHeader);
        processHeaders(responseHeaders);
        CFRelease(responseHeaders);
    }

    numBytes = CFReadStreamRead(httpStream, buf, BUFSIZE);
    handleResponseBodyBytes(myInfo, buf, numBytes);
    break;
}
```

CFHTTPStream—Cleaning Up

```
void cleanUp(CFReadStreamRef httpStream, void *myInfo) {
    // Release system resources
    CFReadStreamClose(httpStream);

    // Guarantee the stream never uses our callback again
    CFReadStreamSetClient(httpStream, 0, NULL, NULL);

    // Get out of the run loop
    CFReadStreamUnscheduleFromRunLoop(httpStream,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);

    // Finally, release our reference on the stream
    CFRelease(httpStream);
}
```

CFHTTPStream—Cleaning Up

```
void cleanUp(CFReadStreamRef httpStream, void *myInfo) {
    // Release system resources
    CFReadStreamClose(httpStream);

    // Guarantee the stream never uses our callback again
    CFReadStreamSetClient(httpStream, 0, NULL, NULL);

    // Get out of the run loop
    CFReadStreamUnscheduleFromRunLoop(httpStream,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);

    // Finally, release our reference on the stream
    CFRelease(httpStream);
}
```

CFHTTPStream—Cleaning Up

```
void cleanUp(CFReadStreamRef httpStream, void *myInfo) {
    // Release system resources
    CFReadStreamClose(httpStream);

    // Guarantee the stream never uses our callback again
    CFReadStreamSetClient(httpStream, 0, NULL, NULL);

    // Get out of the run loop
    CFReadStreamUnscheduleFromRunLoop(httpStream,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);

    // Finally, release our reference on the stream
    CFRelease(httpStream);
}
```

CFHTTPStream—Cleaning Up

```
void cleanUp(CFReadStreamRef httpStream, void *myInfo) {
    // Release system resources
    CFReadStreamClose(httpStream);

    // Guarantee the stream never uses our callback again
    CFReadStreamSetClient(httpStream, 0, NULL, NULL);

    // Get out of the run loop
    CFReadStreamUnscheduleFromRunLoop(httpStream,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);

    // Finally, release our reference on the stream
    CFRelease(httpStream);
}
```

CFHTTPStream—Large Request Body

- Request body may be too big to keep in memory
- Instead, provide a read stream for the body

```
CFReadStreamCreateForStreamedHTTPRequest
    (kCFAllocatorDefault, reqHeaders, bodyStream);
```

- The HTTP stream will open your read stream and pass the bytes through as the request body
- If you provide a Content-Length, we expect it to match the body stream's length
- One catch—no autoredirection!

Network Services

Jeremy Wyld

Zero Configuration Networking

- New in Jaguar
- Local networking protocol
- Easy to use like AppleTalk
- Uses industry standards
- Session 811, Thursday, 2:00pm, Room J

What Does It Provide?

- Standard for advertising services
  - Name-based
  - Service type
  - TCP and UDP
  - Any domain
- Standard for discovering services
  - Domains
  - Services per domain

CFNetServices

- Network Service
  - Represents a single service provider on the network
  - Attributes are domain, type, name, address, and port
- Network Service Browser
  - Represents a search for domains or for services

Using CFNetServices

- Basic usage:
  - Create
  - Schedule
  - Run
  - Handle callbacks

Using CFNetServices

- 3 main tasks
  - Advertise your service
  - Find an existing service
  - Connect to a service you found

Advertising Your Service

```
void  registerMyService(CFStringRef name, void* info) {

    CFStreamError error;
    CFNetServiceContext ctxt =
        {0, info, NULL, NULL, NULL};

    CFNetServiceRef myService =
        CFNetServiceCreate(kCFAllocatorDefault,
            CFSTR(""), CFSTR("_echo._tcp."), name, 7);

    CFNetServiceSetClient(myService, &registerCB, &ctxt);

    CFNetServiceScheduleWithRunLoop(myService,
        CFRunLoopGetCurrent(),
            kCFRunLoopCommonModes);

    CFNetServiceRegister(myService, &error);
}
```

Advertising Your Service

```
void  registerMyService(CFStringRef name, void* info) {

    CFStreamError error;
    CFNetServiceContext ctxt =
        {0, info, NULL, NULL, NULL};

    CFNetServiceRef myService =
        CFNetServiceCreate(kCFAllocatorDefault,
            CFSTR(""), CFSTR("_echo._tcp."), name, 7);

    CFNetServiceSetClient(myService, &registerCB, &ctxt);

    CFNetServiceScheduleWithRunLoop(myService,
        CFRunLoopGetCurrent(),
            kCFRunLoopCommonModes);

    CFNetServiceRegister(myService, &error);
}
```

Advertising Your Service

```
void  registerMyService(CFStringRef name, void* info) {

    CFStreamError error;
    CFNetServiceContext ctxt =
        {0, info, NULL, NULL, NULL};

    CFNetServiceRef myService =
        CFNetServiceCreate(kCFAllocatorDefault,
            CFSTR(""), CFSTR("_echo._tcp."), name, 7);

    CFNetServiceSetClient(myService, &registerCB, &ctxt);

    CFNetServiceScheduleWithRunLoop(myService,
        CFRunLoopGetCurrent(),
            kCFRunLoopCommonModes);

    CFNetServiceRegister(myService, &error);
}
```

Advertising Your Service

```
void registerServiceCB(CFNetServiceRef service,
    CFStreamError* error, void* info) {

    if ((error->domain == kCFStreamErrorDomainNetServices) &&
        (error->error == kCFNetServicesErrorCollision))
    {
        // Formulate a new name.
        ...
        registerMyService(newName, info);
    }

    CFNetServiceSetClient(service, NULL, NULL);

    CFNetServiceUnscheduleFromRunLoop(service,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);

    CFRelease(service);
}
```

Advertising Your Service

```
void registerServiceCB(CFNetServiceRef service,
    CFStreamError* error, void* info) {

    if ((error->domain == kCFStreamErrorDomainNetServices) &&
        (error->error == kCFNetServicesErrorCollision))
    {
        // Formulate a new name.
        ...
        registerMyService(newName, info);
    }

    CFNetServiceSetClient(service, NULL, NULL);

    CFNetServiceUnscheduleFromRunLoop(service,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);

    CFRelease(service);
}
```

Finding an Existing Service

```
void findServices(void* info) {

    CFStreamError error;
    CFMutableArrayRef list = CFArrayCreateMutable
        (kCFAllocatorDefault, 0, &kCFTypeArrayCallBacks);
    CFNetServiceContext ctxt = {0, list, CFRetain,
        CFRelease, CFCopyDescription};

    CFNetServiceBrowser myBrowser =
        CFNetServiceBrowserCreate(kCFAllocatorDefault,
            &myBrowserCB, &ctxt);

    CFNetServiceBrowserScheduleWithRunLoop
        (myBrowser, CFRunLoopGetCurrent(),
            kCFRunLoopCommonModes);

    CFNetServiceBrowserSearchForServices(myBrowser,
        CFSTR(""), CFSTR("_echo._tcp."), &error);
}
```

Finding an Existing Service

```
void findServices(void* info) {

    CFStreamError error;
    CFMutableArrayRef list = CFArrayCreateMutable
        (kCFAllocatorDefault, 0, &kCFTypeArrayCallBacks);
    CFNetServiceContext ctxt = {0, list, CFRetain,
        CFRelease, CFCopyDescription};

    CFNetServiceBrowser myBrowser =
        CFNetServiceBrowserCreate(kCFAllocatorDefault,
            &myBrowserCB, &ctxt);

    CFNetServiceBrowserScheduleWithRunLoop
        (myBrowser, CFRunLoopGetCurrent(),
            kCFRunLoopCommonModes);

    CFNetServiceBrowserSearchForServices(myBrowser,
        CFSTR(""), CFSTR("_echo._tcp."), &error);
}
```

Finding an Existing Service

```
void findServices(void* info) {

    CFStreamError error;
    CFMutableArrayRef list = CFArrayCreateMutable
        (kCFAllocatorDefault, 0, &kCFTypeArrayCallBacks);
    CFNetServiceContext ctxt = {0, list, CFRetain,
        CFRelease, CFCopyDescription};

    CFNetServiceBrowser myBrowser =
        CFNetServiceBrowserCreate(kCFAllocatorDefault,
            &myBrowserCB, &ctxt);

    CFNetServiceBrowserScheduleWithRunLoop
        (myBrowser, CFRunLoopGetCurrent(),
            kCFRunLoopCommonModes);

    CFNetServiceBrowserSearchForServices(myBrowser,
        CFSTR(""), CFSTR("_echo._tcp."), &error);
}
```

Finding an Existing Service

```
void myBrowserCB(CFNetServiceBrowserRef browser,
      CFOptionFlags flags, CFTypeRef service,
      CFStreamError* error, void* info) {

      CFMutableArrayRef list = (CFMutableArrayRef)info;

      if (flags & kCFNetServiceFlagRemove) {
            // Remove the service from the browser list
            CFArrayRemoveValueAtIndex(list, service);

      } else {
            // Add the service to the browser list
            CFArrayAppendValue(list, service);
      }

      if ((flags & kCFNetServiceMoreComing) == 0)
            // Update the UI to reflect the changed services
}
```

Finding an Existing Service

```
void myBrowserCB(CFNetServiceBrowserRef browser,
      CFOptionFlags flags, CFTypeRef service,
      CFStreamError* error, void* info) {

      CFMutableArrayRef list = (CFMutableArrayRef)info;

      if (flags & kCFNetServiceFlagRemove) {
            // Remove the service from the browser list
            CFArrayRemoveValueAtIndex(list, service);

      } else {
            // Add the service to the browser list
            CFArrayAppendValue(list, service);
      }

      if ((flags & kCFNetServiceMoreComing) == 0)
            // Update the UI to reflect the changed services
}
```

Connecting to a Service

```
void  resolveAService(CFStringRef name, void* info) {

    CFStreamError error;
    CFNetServiceContext ctxt =
        {0, info, NULL, NULL, NULL};

    CFNetServiceRef theService =
        CFNetServiceCreate(kCFAllocatorDefault,
            CFSTR(""), CFSTR("_echo._tcp."), name, 0);

    CFNetServiceSetClient(theService, &resolveCB, &ctxt);

    CFNetServiceScheduleWithRunLoop(theService,
        CFRunLoopGetCurrent(),
        kCFRunLoopCommonModes);

    CFNetServiceResolve(theService, &error);
}
```

Connecting to a Service

```
void  resolveAService(CFStringRef name, void* info) {

    CFStreamError error;
    CFNetServiceContext ctxt =
        {0, info, NULL, NULL, NULL};

    CFNetServiceRef theService =
        CFNetServiceCreate(kCFAllocatorDefault,
            CFSTR(""), CFSTR("_echo._tcp."), name, 0);

    CFNetServiceSetClient(theService, &resolveCB, &ctxt);

    CFNetServiceScheduleWithRunLoop(theService,
        CFRunLoopGetCurrent(),
        kCFRunLoopCommonModes);

    CFNetServiceResolve(theService, &error);
}
```

Connecting to a Service

```
void  resolveAService(CFStringRef name, void* info) {

    CFStreamError error;
    CFNetServiceContext ctxt =
        {0, info, NULL, NULL, NULL};

    CFNetServiceRef theService =
        CFNetServiceCreate(kCFAllocatorDefault,
            CFSTR(""), CFSTR("_echo._tcp."), name, 0);

    CFNetServiceSetClient(theService, &resolveCB, &ctxt);

    CFNetServiceScheduleWithRunLoop(theService,
        CFRunLoopGetCurrent(),
        kCFRunLoopCommonModes);

    CFNetServiceResolve(theService, &error);
}
```

Connecting to a Service

```
void resolveCB(CFNetServiceRef service, CFStreamError*
error, void* info) {

    if (error->error == noErr) {

        CFArrayRef allAddrs =
            CFNetServiceGetAddressing(service);

        CFDataRef addr =
            CFArrayGetValueAtIndex(allAddrs, 0);

        connectToAddress(addr, info);
    }
    else {
        ...
    }
}
```

Connecting to a Service

```
void resolveCB(CFNetServiceRef service, CFStreamError*
error, void* info) {

    if (error->error == noErr) {
        …
    }
    else {

        CFNetServiceSetClient(service, NULL, NULL);

        CFNetServiceUnscheduleFromRunLoop(service,
        CFRunLoopGetCurrent(),
        kCFRunLoopCommonModes);

        CFRelease(service);
    }
}
```

Other Threading Models

- Synchronous vs. asynchronous
  - Do not schedule on any run loops
  - Browser still requires a callback
  - Functions will now block
    - CFNetServiceRegister
    - CFNetServiceResolve
    - CFNetServiceBrowserSearchForDomains
    - CFNetServiceBrowserSearchForServices

Demo

Echo client and server

For More Information

- Sample code can be found at
  /Developer/Examples/CFNetwork
- Subscription Mailing List
  macnetworkprog@lists.apple.com

Roadmap

808 Managing I/O: CFRunLoop and CFStream	Room C **Wed., 2:00pm**
811 Zero Configuration Networking: Rendezvous	Room J **Thurs., 2:00pm**

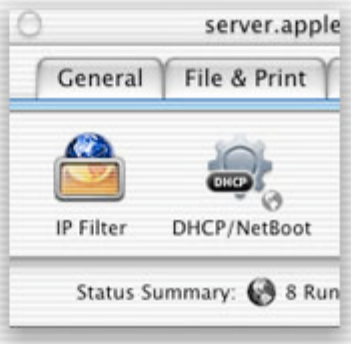
Who to Contact

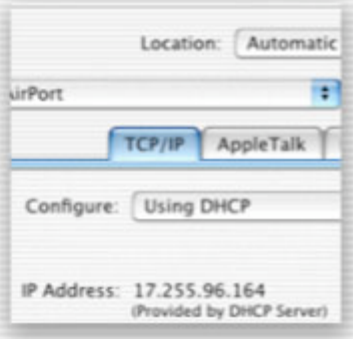
Tom Weyer
Apple Worldwide Developer Relations
weyer@apple.com

Subscription
macnetworkprog@lists.apple.com

http://developer.apple.com/wwdc2002/urls.html

Q&A





Tom Weyer
Apple Worldwide Developer Relations
weyer@apple.com

http://developer.apple.com/wwdc2002/urls.html


WWDC2002


WWDC2002


WWDC2002